

SANDIA REPORT

SAND2015-XXXYY
Unlimited Release
Printed SomeMonth 2015

MueMex User's Guide

Brian Kelley, Christopher M. Siefert

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



SAND2015-XXYY
Unlimited Release
Printed SomeMonth 2015

MueMex User's Guide

Brian Kelley Scalable Algorithms Sandia National Laboratories Mailstop 1318 P.O. Box 5800 Albuquerque, NM 87185-1318 bmkelle@sandia.gov	Christopher M. Siefert Computational Multiphysics Sandia National Laboratories Mailstop 1322 P.O. Box 5800 Albuquerque, NM 87185-1322 csiefer@sandia.gov
---	--

Abstract

This is the user guide for the Matlab interface in MUELU [1].

Acknowledgment

Many people have helped develop the MUELU Matlab interface. We'd like to thank in particular Andrey Prokopenko, Ray Tuminaro, and Tobias Wiesner.

Contents

1	Users	9
1.0.1	Basic Instructions	9
1.0.2	Full Instructions	9
2	Developers	13
2.0.3	Matlab Factories	13
2.0.4	Custom Variables	14
2.0.5	Matlab Callbacks	15
2.0.6	Matlab Callback Example	16
2.0.7	Direct Data Conversion	17
	References	19

Appendix

List of Figures

List of Tables

Chapter 1

Users

Muemex - MueLu Interface for MATLAB

1.0.1 Basic Instructions

1. Run "matlab" script in this directory to launch matlab with proper shared libraries
2. Run "help muelu" from matlab to see detailed help for "muelu" function
3. Basic setup for muelu is "problemID = muelu('setup', A);"
4. Basic solve for muelu is "x = muelu(problemID, b);"
5. Run "ctest" in this directory to run the experimental matlab tests for MueLu

1.0.2 Full Instructions

Navigate to muelu/matlab/bin.

Run MATLAB through the "matlab" script.

Alternatively, find the value for \$LD_PRELOAD printed by the "./matlab" script and set it yourself. Then MATLAB can be run directly with

```
matlab -nosplash -nojvm
```

The "muelu" function will only be available if "muelu/matlab/bin" is in MATLAB's path. If you want to use it from a different directory, run

```
"addpath('path/to/matlab/bin');"
```

from within MATLAB.

With a sparse matrix A, set up a MueLu hierarchy:

```
problemID = muelu('setup', A);
```

Run

```
A = laplacianfun([50, 50]);
```

to get a 50x50 2D Laplace matrix.

Note: any number of problems can be set up at a time. Optionally, pass A and fine level coordinates:

```
problemID = muelu('setup', A, coords);
```

Parameters for the "easy parameter list" are listed after A and coords:

```
problemID = muelu('setup', A, coords, 'coarse: max size', 50);
```

Sublists can be passed using MATLAB cell arrays.

```
problemID = muelu('setup', A, 'level 0', {'aggregation: drop tol', 0.03}, 'level 1', {'aggregation: drop tol', 0.01});
```

XML parameter lists can also be used.

```
problemID = muelu('setup', A, 'xml parameter file', 'myParams.xml');
```

The setup function can also return operator complexity:

```
[problemID, oc] = muelu('setup', A);
```

By default, muemex uses Tpetra objects to store the MueLu preconditioner. This can be customized by setting the parameter "Linear Algebra".

```
problemID = muelu('setup', A, 'Linear Algebra', 'epetra');  
problemID = muelu('setup', A, 'Linear Algebra', 'tpetra');
```

Parameters can be strings, integers, booleans or arrays (full or sparse, real or complex).

Solve a problem with b as the right-hand side vector(s):

```
x = muelu(problemID, b);  
x = muelu(problemID, A, b);
```

The solve function can also return the number of iterations taken by Belos:

```
[x, numIters] = muelu(problemID, b);
```

The first version uses the A that was passed when setting up the problem. The second version uses a new A with the same hierarchy.

Parameters for Belos can be listed after the required parameters:

```
[x, numIters] = muelu(0, b, 'Maximum Iterations', 400);
```

Remove a specific problem and free memory associated with it:

```
muelu('cleanup', problemID);
```

or all problems:

```
muelu('cleanup');
```

List basic information about problem(s):

```
muelu('status', problemID);
muelu('status');
```

Get data from a level:

```
data = muelu('get', problemID, levelID, dataName);
```

This works if dataName is A, P, R, Nullspace, Coordinates, or Aggregates. Otherwise, the data type must also be specified.

```
data = muelu('get', problemID, levelID, dataName, dataType);
```

For example, to get the finest prolongator matrix from problem 0, do

```
P = muelu('get', 0, 1, 'P');
```

Or, to get non-standard data:

```
myList = muelu('get', 0, 0, 'Special NodeList', 'Ordinal Vector');
```

Matrix, MultiVector, OrdinalVector, Int, Scalar, Double and Complex are the supported types. Note that any data retrieved this way must have a keep flag set on it during hierarchy setup. For example, "Aggregates" are not normally available since they are discarded after P is created.

Chapter 2

Developers

Muemex - MueLu interface for MATLAB Developer Features

2.0.3 Matlab Factories

MatlabSmoothen, SingleLevelMatlabFactory and TwoLevelMatlabFactory are implementations of SmootherPrototype, SingleLevelFactoryBase, and TwoLevelFactoryBase that use matlab functions instead of C++ code to generate data in MueLu levels.

Parameters: Set in XML parameter list in the factory instantiation.

MatlabSmoothen: "Needs", "Setup Function", "Solve Function", "Number of Solver Args"

1. "Needs" is a comma-separated list of hierarchy data that the smoother will request, and pass into setup function in order that they are listed.
2. "Setup Function" is the matlab function to run to set up the smoothing. Must take a sparse matrix (A) followed by the matlab types corresponding to "Needs"
3. "Solve Function" is the matlab function to run the solve phase of the smoother. Must take sparse matrix A, double array X, double array B, followed by the outputs of "Setup Function" as arguments. Shouldn't return anything.
4. "Number of Solver Args" (int) is the number of expected outputs of "Setup Function" that will be stored until the solve phase, when they will be passed in after A, x, b.

SingleLevelMatlabFactory: "Needs", "Provides", "Function"

1. "Needs" is list of inputs to the matlab function that will be pulled from level. "Level" is a special key for the Needs list, and will be passed to MATLAB with the current level ID.
2. "Provides" is what will be returned by the matlab function and added to the level.
3. "Function" is the name of the matlab function to run. Parameters/return values must match Needs/Provides.

TwoLevelMatlabFactory: "Needs Fine", "Needs Coarse", "Provides", "Function"

1. Just like SingleLevelMatlabFactory, but inputs come from both fine and coarse levels.

2.0.4 Custom Variables

Muemex also supports setting custom data in the hierarchy. To use this feature, set the data you want as a parameter in a level sublist when setting up the problem. For example, if you want a matrix "MyMatrix" to be available to a matlab factory, set up with this command:

```
muelu('setup', A, 'level 0', {'Matrix MyMatrix', MyMatrix}, 'xml parameter file', 'myXMLParams.xml');
```

Notice the type specifier "Matrix" before the variable name. This is part of the name, and has to be included in every mention of the variable. The type is not case sensitive; 'MultiVector' and 'multivector' are equivalent. The name itself can be any string without whitespace. Names have to unique within their level - there can't be "Double d1" and "Int d1".

If an aggregates factory needs that custom matrix, the XML parameter list for the problem might look like this:

```
...
<Parameter name="factory" type="string" value="SingleLevelMatlabFactory"/>
<Parameter name="Provides" type="string" value="Aggregates"/>
<Parameter name="Needs" type="string" value="A, Matrix MyMatrix"/>
<Parameter name="Function" type="string" value="simpleAggregation"/>
...
```

This will send MyMatrix to simpleAggregation as the second argument. There must always be exactly one space between the type and the name. Leading and trailing spaces are always ignored.

The following custom variable types are supported:

1. Matrix (sparse, MxM, real or complex depending on MueLu context)
2. MultiVector
3. OrdinalVector (must be Mx1 column vector of int32)
4. Int (32 bit signed)
5. Scalar (double or complex depending on context)
6. Double
7. Complex

Note: Custom variables added to the hierarchy either through muelu('setup') or by a matlab factory are never removed from their level - a UserData keep flag is set for them.

2.0.5 Matlab Callbacks

The "MueLu_MatlabUtils" code contains the backend for the matlab factories. There is a system for defining inputs and outputs for arbitrary matlab functions. There are also functions for converting between matlab data types (mxArray*) and C++/Xpetra data types.

```
vector<RCP<MuemexArg>> callMatlab(string function, int numOutputs, vector<RCP<
    MuemexArg>> args);
```

This function is declared in "MueLu_MatlabUtils_decl.hpp". It calls the matlab function "function", with "args" as arguments, expecting numOutputs outputs, and returning the list of returned values. The MuemexArg class is a basic wrapper for the several types of data that can be passed to and from matlab. It only stores a MUEMEX_TYPE representing the underlying type of the data:

```
enum MUEMEX_TYPE
{
    INT,
    DOUBLE,
    STRING,
    COMPLEX,
    XPETRA_ORDINAL_VECTOR,
    TPETRA_MULTIVECTOR_DOUBLE,
    TPETRA_MULTIVECTOR_COMPLEX,
    TPETRA_MATRIX_DOUBLE,
    TPETRA_MATRIX_COMPLEX,
    XPETRA_MATRIX_DOUBLE,
    XPETRA_MATRIX_COMPLEX,
    XPETRA_MULTIVECTOR_DOUBLE,
    XPETRA_MULTIVECTOR_COMPLEX,
    EPETRA_CRSMATRIX,
    EPETRA_MULTIVECTOR,
    AGGREGATES,
    AMALGAMATION_INFO
};

class MuemexArg
{
    MuemexArg(MUEMEX_TYPE dataType);
    MUEMEX_TYPE type;
};
```

MuemexData is a subclass of MuemexArg. It has one the types listed in MUEMEX_TYPE as a field.

```

template<typename T>
class MuemexData
{
public:
    MuemexData(T& data);
    MuemexData(const mxArray* mxa);
    mxArray* convertToMatlab();
    T& getData();
private:
    T data;
};

```

MuemexData can either be constructed using the actual underlying data object, or an mxArray*. Either way, the data is copied and stored as a C++/Xpetra object. Note that MuemexData can only be used with Trilinos types wrapped in Teuchos::RCP. Int, double, string and complex should not be wrapped.

Any RCP<MuemexData<T>> object can be converted to a pure RCP<MuemexArg> with

```
RCP<MuemexArg> arg = rcp_implicit_cast<MuemexArg>(mmData);
```

and back with

```
RCP<MuemexData<T>> mmData = rcp_static_cast<MuemexData<T>>(arg);
```

The type of the underlying MuemexData data is stored in MuemexArg::type.

2.0.6 Matlab Callback Example

This matlab function calculates the inverse tangent and also times the calculation.

```

function [angle, timeElapsed] = timedAtan(y, x)
    tic;
    angle = atan2(y, x);
    timeElapsed = toc;
end

```

To use the function from C++, do:

```

double y = 5.423;
double x = 42.6;
vector<MuemexArg> inputs;
inputs.push_back(rcp(new MuemexData<double>(y)));
//add the first argument (y)

```

```

inputs.push_back(rcp(new MuemexData<double>(x)));
                           //add the second argument (x)
/*
Note that it is not necessary to rcp_implicit_cast the RCP<MuemexData>
objects when adding them to MuemexArg vector - this happens implicitly
*/
vector<MuemexArg> outputs = callMatlab("timedAtan", 2, inputs);
                           //call the function, expect 2 outputs
RCP<MuemexData<double>> angleOutput = rcp_static_cast<MuemexData<double>>(
    outputs[0]); //recover the outputs, knowing both have type "double"
RCP<MuemexData<double>> timeOutput = rcp_static_cast<MuemexData<double>>(
    outputs[1]);
printf("The arctan of %f/%f is %f, and the calculation took %f seconds.", y, x
, angleOutput->getData(), timeOutput->getData());

```

2.0.7 Direct Data Conversion

It is also possible to directly convert between matlab arrays and C++ objects (without using a MuemexData wrapper). To access a matlab array pointer "mxa" from C++, do:

```
int myInt = loadDataFromMatlab<int>(mxa);
```

To copy a C++ object to matlab, do:

```
mxArray* mxa = saveDataToMatlab(data);
```

The following types are supported by these functions ('Node' is the default Kokkos serial node type):

1. int
2. double
3. std::complex<double>
4. std::string
5. RCP<Xpetra::Matrix<double, int, int, Node>>
6. RCP<Xpetra::Matrix<complex, int, int, Node>>
7. RCP<Xpetra::MultiVector<double, int, int, Node>>
8. RCP<Xpetra::MultiVector<complex, int, int, Node>>
9. RCP<Tpetra::CrsMatrix<double, int, int, Node>>

10. `RCP<Tpetra::CrsMatrix<complex, int, int, Node>>`
11. `RCP<Tpetra::MultiVector<double, int, int, Node>>`
12. `RCP<Tpetra::MultiVector<complex, int, int, Node>>`
13. `RCP<Epetra_CrsMatrix>`
14. `RCP<Epetra_MultiVector>`
15. `RCP<Xpetra::Vector<int, int, int, Node>>`
16. `RCP<MueLu::Aggregates<int, int, Node>>`

Note that the Aggregates structure in matlab must be constructed with the "constructAggregates.m" function:

```
function agg = constructAggregates(nVertices, nAggregates, vertexToAggID,
    rootNodes, aggSizes);

    % nVertices: Total number of nodes in the problem (since all MueMex
    % problems are serial)
    % nAggregates: Total number of aggregates
    % vertexToAggID: Array of length nVertices, maps nodes to aggregate IDs
    % rootNodes: Array of length nAggregates, contains node IDs of all the
    % aggregate roots
    % aggSizes: Array of length nAggregates, contains the number of nodes
    % in each aggregate
```

References

- [1] Jonathan J. Hu, Andrey Prokopenko, Christopher Siefert, and Raymond S. Tuminaro. MueLu multigrid framework. <http://trilinos.org/packages/muelu>, 2014.

DISTRIBUTION:

1 Tobias Wiesner
Institute for Computational Mechanics
Technische Universität München
Boltzmanstraße 15
85747 Garching, Germany

1 MS 1320	Michael Heroux, 1446
1 MS 1318	Robert Hoekstra, 1446
1 MS 1320	Mark Hoemmen, 1446
1 MS 1320	Paul Lin, 1446
1 MS 1318	Andrey Prokopenko, 1426
1 MS 1322	Christopher Siefert, 1443
1 MS 0899	Technical Library, 9536 (electronic copy)

DISTRIBUTION:

- | | |
|-----------|---|
| 1 MS 9159 | Jonathan Hu, 1426 |
| 1 MS 9159 | Paul Tsuji, 1442 |
| 1 MS 9159 | Raymond Tuminaro, 1442 |
| 1 MS 0899 | Technical Library, 8944 (electronic copy) |



Sandia National Laboratories